RibTools Developer Manual

Author: Davide Pasca
Version: 2010/6/23

Contents

Preface

This software is released under the New BSD license.

The code is available at http://ribtools.googlecode.com .

See the User Manual for instructions on how to execute *RibTools* binaries.

Requirements

RibTools currently only compiles for Windows, although the code is written with portability in mind and Windows-specific code is kept to a minimum.

On Windows the required tools are:

- Microsoft Visual Studio 2008 SP1
- CMake 2.8 or better

Directory Structure

As obtained from the repository:

Source/

DImage/	Bitmap images library
DispDrivers/	Display drivers sources
DistribSrc/	Source assets
DMath/	Math library sources and project
docs/	Documentation related files (currently just some images)
DSystem/	Base system library sources and projects
external/	External libraries, directly included
externals/	External libraries included automatically by SVN
RI_System/	Core rendering engine and RenderMan-interface
RibRender/	RibRender application project and sources
RibRenderLib/	A library that groups most functionalities used by both *RibRender* and *RibRenderToy*
RibRenderServer/	RibRenderServer application project and sources
RibRenderToy/	RibRenderToy application project and sources

RSLCompilerCmd/	RSLCompilerCmd application project and sources
RSLCompilerLib/	RSL compiler library
CMakeLists.txt	Root CMake file to create the build files
license.txt	License file
make_build.bat	Script that creates the `../build` directory and its contents (Windows)
make_build.sh	Script that creates the `../build` directory and its contents (Linux/OSX)
make_distrib.bat	Script that creates the `../Distrib` directory and its contents (Windows)
make_distrib.sh	Script that creates the `../Distrib` directory and its contents (Linux/OSX)
make_install.bat	Script that builds a .zip file for binary distribution (assumes EXEs have been manually compiled with VS)
readme.txt	Readme file in txt format (and reST)
readme.html	Readme file in HTML format

The Distrib Directory

A directory named `Distrib` is generated outside `Source` as part of the build process. This directory is automatically generated, and it can therefore be safely erased (unless one has any important changes in it !)

The `Distrib` directory contains both data coming from `DistribSrc` and executables generated when compiling with Visual Studio (see below).

When running, it's is advised to have the environment variable `RIBTOOLS_DIR` set to the `Distrib` directory. For more informations on how to setup the environment variable, see the instructions at the *General Usage* section of the User Manual.

Compiling and running from VS

Building the solution

From the *Source* directory launch the batch `make_build.bat` This script will create he `build` directory and automatically call `make_distrib.bat` to create the `Distrib` directory.

Launch the newly created Visual Studio solution that can be found at `build/RibTools.sln`.

From the Standard Toolbar select a *solution configuration* (**Debug** or **Release**).

Build the solution by selecting the option from the **Build** menu (or press F7 or Ctrl+Alt+B, depending on the VS configuration)

Once the build is finished the `Distrib` directory should contain 4 executables:

- *RibRender.exe*
- *RibRenderServer.exe*
- *RibRenderToy.exe*
- *RSLCompilerCmd.exe*

Note that for every *.exe* file there is a corresponding *.idb* file. These files hold debug data necessary to display symbols when debugging with Visual Studio's debugger.

A little test

- From the `Distrib` directory, launch `RibRenderToy.exe`
- Right-click inside the RibRenderToy window and select a test scene from the pop-up menu (*Airplane.rib* is a safe bet)

- After a little while the scene should appear in all it's beauty (!)

Running from Visual Studio

In order to run form visual studio, a one time setup is necessary.

Set the choosen application as the StartUp project

Right-click on an executable project (e.g. *RibRenderToy*), and choose **Set as StartUp Project** from the pop-up menu.

Setup the working directory

- Right-click on the application project (e.g. *RibRenderToy*) and select **Properties** from the pop-up menu
- From the list at the left of the dialog, select **Configuration Properties -> Debugging** - Where it says **Working Directory** set it to `..\..\Distrib`

Note: Set this for all the configurations (*Release* and *Debug*). A common mistake is to set up the *Working Directory* only for the current configuration and forget about the other configurations.

Run (!!!)

Run the application with or without the debugger by pressing F5, Ctrl+F5 (or whatever is your configuration 8)

Code modules overview

Code modules are subdivided by directories inside the `Source` directory.

DImage

DImage handles bitmap images. It also interfaces with PNG, JPEG and TIFF readers available in the `externals` directory.

DispDrivers

This module provides interfaces for the rendering output, here called as *display drivers*.

There are currently only two display drivers. One provides output to an OpenGL texture, for immediate display. The other driver stores the rendering result to an image file (PNG, JPEG or TIFF).

Display drivers are strictly correlated to the `Display` RIB command.

DMath

This module provides basic functionalities for linear algebra and random numbers generation.

DMath is developed with SIMD or vector processors in mind. Base implementations of SIMD-fied vectors are implemented for the SSE2 instruction set (utilizing compiler intrinsics).

Additional non-SIMD and LRBni implementations are provided for compatibility and as a proff of concept.

DSystem

This module provides various low-level services, such as memory allocation and threading, generic I/O and network interface.

In many cases *DSystem* only provides a thin wrap around the OS (e.g. memory management and file I/O).

RI_System

This module is the heart of the renderer. *RI_System* implements a RenderMan-like state machine, and commands the whole rendering pipeline to the final production of rendered pixel for the *display drivers* (see above).

RibRender

This is the basic rendering application that takes a RIB file in input and produces a rendering to an output described in the RIB file.

RibRenderServer

This is an application that waits for incoming TCP/IP network connections from an instance of *RibRender* and renders sections of an image on-demand.

RibRenderServer is a pure console application and it doesn't produce any rendering output by itself.

RibRenderToy

This is an interactive application based on the *freeglut* library for interface and OpenGL for display.

RibRenderToy is meant to be used with a mouse for quick visual testing.

RibRenderLib

This library implements many of the functionalities necessary for *RibRender*, *RibRenderToy* and *RibRenderServer*.

RSLCompilerCmd

This is an application that compiles a RenderMan-like shader source (`sl` file) into a custom `rrasm` file, the native format of the *RibTools*' shader virtual machine.

This command is provided for debugging purposes only, as rendering applications such as *RibRender* will automatically compile shaders as necessary.

RSLCompilerLib

This library is utilized directly by the rendering applications such as *RibRender* to compile the RenderMan-like shading language into `rrasm`.

It is also obviously used by *RSLCompilerCmd*.